Enhancing Adaptive Middleware for Quantum Chemistry Applications with a Database Framework

Lakshminarasimhan Seshagiri,
Meng-Shiou Wu, Masha Sosonkina
Scalable Computing Laboratory
Ames Laboratory
Ames, IA 50011, USA
Email: sln,mswu,masha@scl.ameslab.gov

Zhao Zhang
Department of Electrical
and Computer Engineering,
Iowa State University,
Ames, IA 50011, USA
Email: zzhang@iastate.edu

Mark S. Gordon,
Michael W. Schmidt
Department of Chemistry
and Ames Laboratory,
Iowa State University,
Ames, IA 50011, USA
Email: mark,mike@si.msg.chem.iastate.edu

***Abstract*— Quantum chemistry applications such as the General Atomic and Molecular Electronic Structure System (GAMESS) that can execute on a complex peta-scale parallel computing environment has a large number of input parameters that affect the overall performance. The application characteristics vary according to the input parameters. This is due to the difference in the usage of resources like network bandwidth, I/O and main memory, according to the input parameters. Effective execution of applications in a parallel computing environment that share such resources require some sort of adaptive mechanism to enable efficient usage of these resources. In our previous work, we have integrated GAMESS with an adaptive middleware NICAN (Network Information Conveyer and Application Notification) for dynamic adaptations during heavy load conditions that modify execution of GAMESS computations on a per-iteration basis. This leads to better application performance. In this research, we have expanded the structure of NICAN in order to include other input parameters based on which application performance can be controlled. The application performance has been analyzed on different architectures and a tuning strategy has been identified. A generic database framework has been incorporated in the existing NICAN mechanism so as to aid this tuning strategy.**

I. Introduction

GAMESS [12] is one of the applications used by chemists worldwide to perform *ab initio* calculations. MOLPRO [21], NWChem[7], MPQC[6] , Q-Chem [25], Psi3 [26] and MOLCAS [27] are some of the other packages that are used for performing *ab initio* calculations. Computational chemistry applications such as GAMESS are widely used to perform *ab initio* molecular quantum chemistry calculations. These calculations include a wide range of Hartree-Fock(HF) wave function calculations such as RHF(Restricted Hartree-Fock) for species with all electrons paired (closed shells), ROHF(restricted open shell Hartree-Fock) and UHF(unrestricted Hartree Fock) for open shell species, GVB(Generalized valence bond wavefunction) and MCSCF(Multiconfigurational SCF wavefunction). The capabilities of GAMESS has been described in detail in [12] and in the available GAMESS distribution. Using the Hartree-Fock self consistent field (SCF) method, GAMESS iteratively approximates a solution to the Schrödinger equation that describes the basic structure of atoms and molecules. The SCF method has two implementations, *direct* and *conventional*, which differ from each other in the handling of the two-electron (*2-e*) integrals. In the *conventional* SCF method, the 2-e integrals are calculated once at the beginning of the SCF process and stored in a file on disk for subsequent iterations. In the *direct* SCF method, the 2-e integrals are recalculated for each iteration.

There are many different approaches to tuning high performance chemistry applications like compiler based optimizations, performance modeling and adaptive algorithms. Dongarra and Eijkhout [1] discuss about *Self-adapting Numerical Software* systems that consist of a framework to automatically pick the best software/hardware combination for high performance computing. Li et al. [10] have described a component based method that enables CQoS(Computation Quality of Service) through a generic database component. This component interacts with different chemistry components to provide an adaptive mechanism that achieves better performance than any trial and error approach. Ustemirov et al. use [22] a middleware tool NICAN to perform this adaptation in GAMESS. We obtained some GAMESS performance data and analyzed part of this data in our work *Exploring Tuning Strategies for Quantum Chemistry Applications* [15]. In this work, we have utilized these analysis results, created a performance database to store this data, devised a database assisted tuning strategy and incorporated it into the NICAN functionality. The results obtained by implementing this tuning strategy have been presented in this paper.

The rest of the paper is organized as follows. Section II provides an introduction to the integration of the NICAN library with GAMESS. Section III looks at the methodology followed to obtain the performance benchmarking results on different architectures. The tuning strategy formulated based on the GAMESS performance testing has been described in Section IV. Section V showcases the database framework modifications to NICAN and presents the results obtained using the modified adaptation mechanism. Finally we talk about the future direction that this work can take in Section VI.

II. GAMESS-NICAN Integration

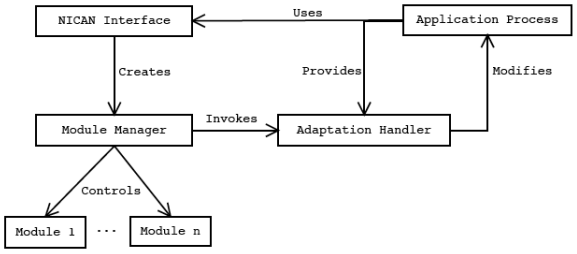

Fig. 1. NICAN layout

Network Information Conveyer and Application Notification or NICAN [19] is a framework that is used to provide methods for applications to adapt their utilization of computational resources based on various conditions. NICAN has been successfully integrated with several applications to aid their execution in a distributed environment. The adaptation in GAMESS using NICAN was designed for SMP (Symmetric Multi Processor) clusters in order to improve GAMESS performance [22] and it focused on the SCF algorithm. Selection of the correct electronic structure calculation routine has a very big effect on the overall computation time. The iterative nature of the SCF algorithm allows us to switch between the *conventional* and *direct* implementations in an arbitrary SCF iteration. The switching is carried out using NICAN in order to decouple the application from having to make any adaptation decisions during its execution. The application is responsible only for the invocation of the adaptation handlers. The adaptations are handled by a control port that is part of the NICAN tool. The adaptation scheme consists of a static and a dynamic part. Every *conventional* GAMESS job gets modified to a *direct* execution mode if there exists a "peer" *conventional* GAMESS job already running in the system [22], [23]. This constitutes the static adaptation method. During the dynamic adaptation phase, the control port gathers system and application information during the iterative SCF calculations and decides on the adaptation at runtime. The algorithm has been explained in [22] and [14]. The experimental results obtained for this algorithm on a SMP have been given in [22]. It has been shown on a two processor system with I/O congestion, that the performance of dynamically adaptive GAMESS is nearly the same as a "no-congestion" case. If the I/O bandwidth is fully consumed, then the adaptation scheme gives a two time improvement in the execution time of GAMESS. Also, on running two simultaneous parallel GAMESS jobs on two and four processors, a gain of 10-15 percent in the cumulative execution time is obtained through the dynamic adaptation scheme.

III. Methodology

We have observed in the works by Ustemirov, Sosonkina et al. ([22]) and Li, Kenny et al[10] that the adaptation algorithm depends only on the wall clock time. In [22], the data regarding the iteration time is collected on-the-fly and utilized by the middleware NICAN in order to make the adaptation decision. In [10], the data is collected offline and fed into a database which helps make a decision on tuning the quantum chemistry application. One of the disadvantages of using only coarse grained performance data is that it prevents us from gaining insight into how and why a computation performs differently on different architectures, and why different sets of molecules can show totally different performance characteristics. We can get a better understanding of these issues if we profile the application on different architectures allowing us to analyze the application performance thoroughly. This would also help us to improve existing tuning strategies. To design tuning strategies for large parameter set applications such as GAMESS, a methodology that spans data acquisition, performance data, metadata management and performance analysis is desired. We proceeded to design this tuning strategy by following a collection-analysis-implementation method. The performance data for GAMESS was collected on different architectures and using different sets of molecules so as to understand the application performance variation. Using this data, application performance analysis was performed and the performance trends were identified. An adaptation strategy was formulated depending on these trends.

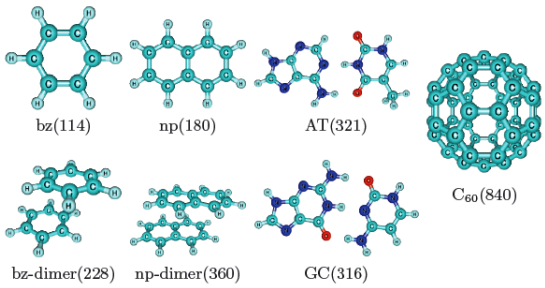

Fig. 2. Molecules Used

In order to obtain the performance data for analysis, we chose two different sets of molecules for testing. One set of molecules has a varying molecular structure that allows us to analyze the performance characteristics as the molecule size varies. These molecules were Benzene(bz) and its dimer, Naphthalene (np) and its dimer (np-dimer), Adenine-Thymine DNA base pair (AT), Guanine-Cytosine DNA base pair(GC) and Buckminsterfullerene (C60). The second set of molecules (Picene, Pentacene, Dibenz Anthracene (J and H) , Benzo naphthacene and Benzo triphenylene) were very similar to each other in structure and allow us to see the performance variation for minute molecular changes. These tests were conducted on three different architectures. One was an Ames Laboratory SMP cluster "Borges", consisting of 4 nodes, each node having two dual-core 2.0GHZ Xeon Woodcrest"

CPUs. The second was the NERSC Franklin cluster with each of Franklin's compute nodes consisting of a 2.3 GHz single socket quad-core AMD Opteron Budapest" processor. The third architecture was a stand alone machine running on the Sun Niagara T2 processor [20], [8]. The T2 processor has a unique architecture that consists of 8 SPARC physical processor cores built in a single chip and each core is capable of running 8 threads. The application profiling was done using the TAU [16] (Tuning and Analysis Utility) toolkit, which is a popular multi-level and multi-user source code profiler and instrumentor. Using TAU, we split the application run time into the time spent by the application in computing the required data, the time spent by different threads of the program communicating with each other and the time spent by the program in moving the data back and forth between the disk and the memory (the time spent in I/O). These three components usually provide a good insight as to where the application is getting slowed down during its execution. Since usage of the TAU profiler may slow down the application considerably, the profiler was operated on functions that can be broadly classified as ones which provide communication, IO and computation. The profiler output data for each of these functions invoked by GAMESS, were collated to obtain a value for the total time spent on communication, IO or computation. Further details regarding the methodology used has been provided in [15].In this paper, we have provided the scalability results only for the Franklin cluster and the Niagara system.

IV. Performance Analysis and Tuning Strategy

Some of the performance analysis has been published in [15]. We summarize those performance results below, give new analysis data and then show how the tuning strategy was formulated on the basis of these results. The tests were conducted for different input combinations of the two molecule sets. We concentrated on combining the *direct* and *conventional* implementations with HF-SCF computations (Skip the electron correction) and MP2 computations (Möller-Plesset 2^{nd} order energy correction, increased output accuracy) for 13 molecules. On each architecture, we performed at least 8 different test runs for different combinations of input processors and nodes. The performance tool split the wall clock time as communication time, computation time and I/O time. This allowed us to broadly generalize the performance trends for the selected molecules and the input parameters. We obtained 96 different sets (molecule name, architecture, SCF type, Order of perturbation) of performance data for a single molecule. The data sets would increase even further if we get the data for other SCF implementations like UHF and ROHF or if we modify the run type from energy to optimize, gradient etc. The challenge is to ensure that this data is in an easily analyzable format to discover the performance issues and decide on strategies for consistent good performance. Also, all the performance data should be stored in a manner which allows easy data management. Hence it was decided that a database containing the relevant data would be integrated with GAMESS-NICAN.

The direct and conventional methods of execution for GAMESS differ in the way that the SCF computations are performed. Since the direct implementation uses the available memory for computations, it is intuitive that the computation time for a direct implementation would take the most amount of time while the I/O time would be negligible. On the other hand, the conventional implementation time would have some I/O component. Also, for most implementations, it can be deduced that the conventional implementation would be faster since getting the integral data from the files would be faster than recalculating it. However, for larger molecules, it has been shown in [23] that the conventional execution slows down if there is a high IO usage in the system. This difference in the execution time of conventional and direct implementations is exploited in [22] to obtain an adaptation mechanism. Other GAMESS implementation trade-offs are possible as long as they don't compromise the desired accuracy or theory level. For example, for the MP2 level of theory the computation time is very high but HF-SCF, which is a lower level of theory, may not be substituted for MP2 as a means of adaptation to the environment. Instead, one may focus on running MP2 more efficiently and use the adaptation mechanism as a check on the amount of memory requirement to run the job as stated by the user in the input file. The results on the three architectures have shown that specific input node and processor combinations have different characteristics. Having this data enables us to decide the best possible input node-processor combination for running the application in order to obtain the optimum performance. Detailed analysis of the performance data has been published in our work [15].

The computational effort of *ab initio* methods scales formally as at least N^4 where N is the number of basis functions describing the atom. While expanding an unknown function such as a molecular orbital, into a set of known functions, we would require an infinite number of such functions. These represent the complete basis for the unknown function. However, this is not possible and we end up using a finite basis to represent the unknown function. For finite basis, only the components of the molecular orbitals along coordinate axis corresponding to selected bases can be represented. A smaller basis gives a poorer representation and affects the accuracy of the *ab initio* methods. Thus, a balance between accuracy of solution and computational efficiency can be achieved by the proper choice of the basis set [5]. For all the performance tests conducted and represented in [15] and in this paper, we have used a Dunning-type correlation consistent (cc) basis set [12], [5] for a specific hierarchy level (polarized valence double Zeta). There are a total of 391 published basis sets as given in the Basis Set Exchange [3], [13]. For each basis set type, the number of gaussian basis functions and basis set shells vary and this variation

TABLE I

BASIS SET DETAILS (CCD) FOR DIFFERENT MOLECULES

Molecule	np-dimer	C60	AT
Basis Set Shells	168	360	174
Cartesian Gaussian Basis Fns	380	900	400
Molecular Orbitals	360	840	321

can be used as a means of comparing different molecules. The table I shows the basis set functions and shells(Cartesian gaussian basis function groupings) for the molecules np-dimer($(C_{10}H_8)_2$), C60(C_{60}) and AT($C_5N_5H_5 \cdot C_6N_2O_2H_6$).

We can see the correlation between the number of Basis set shells and Cartesian Gaussian Basis functions to the performance of the individual molecules (np-dimer and C60). The performance of these two molecules has been detailed in [15]. As the basis set shells and functions increase, the runtime of the application for a molecule increases, and the computational complexity also increases. The scaling factor of 4 increases even further for post-Hartree-Fock computations (MP2 scales as N^5) as the number of basis functions increase. This is evident from the performance values obtained for MP2 computations. For the purposes of this study it is assumed that two molecules with the same number of basis functions will have approximately the same 2-e integral demands, even though in reality this is not strictly true. The basis set for a given molecule allows us to analyze the computational requirements and performance for that molecule. It may be considered as a useful tool to extrapolate to the performance of a molecule given the basis set parameters from any other molecule whose performance characteristics are known.

V. DATABASE ASSISTED ADAPTATION AND RESULTS

There were two separate issues associated with the performance data that was generated. The data management is a huge part of the problem mainly due to the amount of performance data generated. We would require different analysis methods and programs in order to obtain all relevant information related to the application and system performance. This analysis should help us to devise new rules for the GAMESS-NICAN adaptation mechanism. The adaptation architecture was devised with these two objectives in mind.

A. Adaptation Architecture

Figure 3 shows the complete architecture of the database adaptation framework. It is important to note that the data management as well as the adaptation are considered as part of this framework. The architecture can be divided into two distinct sections. In the offline "Performance Data Collection and Analysis" section, we instrument the GAMESS code using TAU, collect the performance data for different combinations of the application and system parameters and store the data into a *PostGreSql* [2] performance database. This data is then analyzed by separate programs to obtain such metrics like scalability on a particular machine. The second section of the architecture is the "Decision and Activation" Stage which involves the application adaptation using the NICAN middleware tool.

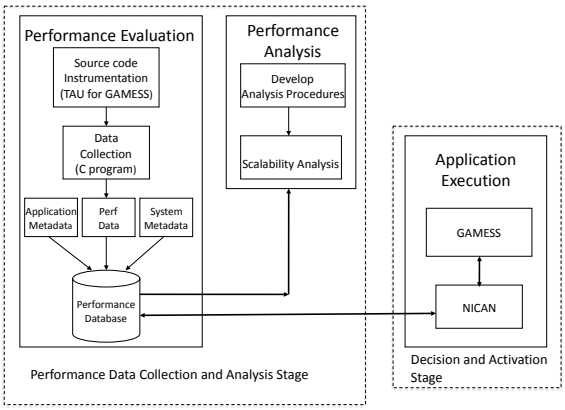

Fig. 3. Database Adaptation Architecture

B. Adapatation Strategy

The amount of data generated for the 13 molecules on the three different architectures for different input combinations clearly showed a need for using full fledged performance scenarios in the adaptation process. We chose *PostGreSql* database for storing the application metadata and the adaptation related data. The application connects to the database using a database daemon server (DBd) that is started by the application. The application ensures that only a single DBd is running for that particular machine, since we have a single database instance for the entire machine. The architecture is similar to a client-server architecture wherein the different application jobs running on the machine are the clients connecting to a single DBd server. The user provides the database connection string to the application through the existing NICAN XML file. This enables each user to connect to his own schema in the performance database. The database stores machine specific details like the cluster node names, node configurations, available memory and other performance related details such as the best node-processor combination. The data format of the data exchanged between the DBd and the NICAN module is predefined. The database class provided to the NICAN module contains all the functions required to extract the result from the database. One other advantage of using a DBd server is that different modules of the same application job can connect to the database at the same time. Currently, the database management is handled manually and the data for each molecule is inserted by the programmer.

The tuning strategy that we propose on the basis of the results obtained augments the existing NICAN adaptation strategy. The current implementation of NICAN requires a check run in order to get the memory requirements of the input molecule. We have offloaded this information into the database. The amount of data being written by GAMESS into files depends on the input parameters chosen. For example, when the *gbasis* value is taken as CCQ (Dunning-type Correlation Consistent polarized Valence Quadruple Zeta) , the amount of data written for the *conventional* method is very large. The external disk sizes on clusters are normally huge. However, it is possible that in case of small clusters, the disk size might get exceeded due to residual files. Hence, NICAN calculates the amount of space available to store integral files and in case sufficient space is not available, the implementation can be modified to *direct*. In our work [15], we showed that MP2 computations for large molecules such as C60 fail if the required shared memory is not available on every node for constructing the required matrices. Thus, the shared memory availability is a very important factor in the success of a MP2 computation. It is not possible for us to change the shared memory on every node depending on the requirement of every single job. The solution would be to distribute the job on multiple nodes and provide the required memory to the GAMESS job. NICAN compares the memory requirement and the memory requested by the user. The job is immediately stopped if the memory requested by the user is less than the memory required to execute the job. The correct value of DDI memory is printed in the log file. This ensures that the job is stopped before the start of a very expensive execution instead of producing failure at the MP2 calculation stage.

From the performance analysis, we have seen that for a given number of application processes on a given architecture, the most efficient application performance on a given architecture is obtained at a specific distribution of GAMESS processes per core/node. Such combinations are stored in the database for each particular operating environment. For example, on a Sun T2 Niagara machine, the best method to obtain fastest application run time would be to distribute the number of GAMESS processes to as many cores as possible. This approach may not guarantee the best performance for all scenarios though. Obviously, this adaptation would be possible only if there are cores or nodes (in case of Franklin and Borges) available so as to distribute the processes. We can get a decent speed up for large molecules by increasing the number of cores used for execution on the Niagara machine. The adaptation between *conventional* to *direct* is an existing feature in NICAN and we have not modified the algorithm. The database framework has been incorporated as an easy extensibility to the NICAN features.

C. Database framework adaptation results

The database modification to the NICAN adaptation was tested on Borges using two representative molecules, AT and C60, and for SCF computations. The database was created on Borges and the necessary data was inserted into this database. The results are shown in Figure 4. It is possible to show the performance gain of the database framework by using a single molecule. But the test run would not be able to take advantage of the static adaptation. The Y-axis of the graph shows the combined execution time for both these molecules. The X-axis of the graph represents the input node-processor combination. We can see that the performance improvement varies from around 28% to 54%. This improvement is mainly due to offloading the idealized iteration time to the database instead of computing it every time the adaptation has to run.

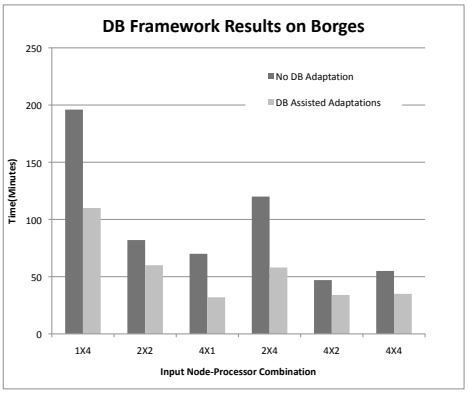

Fig. 4. Database Adaptation Results

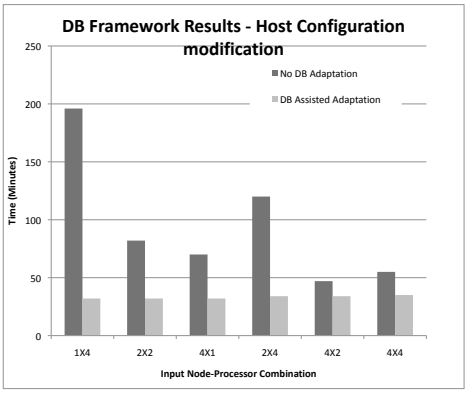

Fig. 5. Database Adaptation Results with host configuration modification

The host configuration modification would ensure that for a specific number of processes, the input node-processor configuration is always changed to a particular value. For example, if the user requests 4 processes, the best performance is obtained when the job is run on 4 nodes with each node running a single process. The results for host configuration modification are shown in Figure 5. We have also tested the other capabilities

added to NICAN. The GAMESS adaptation was tested when the filesystem limit was reached. The filesystem limit was kept at 95% and the GAMESS execution was modified from *conventional* to *direct* when this filesystem limit was reached. The NICAN library was modified to check the database for MP2 memory requirements and then compare it with the memory requested by the user. The job is appropriately terminated if the memory requirements do not match.

D. Database Framework Scalability Analysis

The performance data collected can be utilized for different analyses and derive a variety of analysis results. One advantage of having a vast amount of performance data for an application such as GAMESS is to be able to analyze the scalability of the application on different architectures and deduce the inflection point for the performance degradation or for performance improvement. Our scalability analysis indicates the improvement in the performance of GAMESS on a particular architecture, when the number of cores or nodes, on which the application is executed, is increased. We have shown the scalability analysis of GAMESS on Franklin and Niagara processors below.

The scalability of GAMESS on the different architectures is calculated using a separate C program. This program operates on the TAU results file; takes the molecule name and SCF type as input and outputs the scalability results in tabular format. Scalability is calculated as the *inverse* ratio of the total time taken by GAMESS on a given number of nodes and the total time taken by GAMESS for a single node, when the number of GAMESS processes per node is constant. In other words, we are defining a strong scalability for GAMESS as we try to plot the variation in the solution time with the number of processors for a fixed problem size. A node can contain multiple cores and hence is capable of running multiple processes. The tables II, III, IV and V show SCF for the molecule AT. The tables VI and VII show the MP2 scalability for the molecule AT. The 'X' values indicate that results were not obtained for those particular combinations. Consider the following example to understand the scalability calculation. Table IV shows the scalability of a conventional GAMESS job on Franklin. As per our definition, the scalability while using 16 nodes and a single process on each node, would be the *inverse* ratio of total time obtained by running a single process of GAMESS on 16 nodes each and the total time obtained by running a single process of GAMESS on a single node. Since there are no results available for the denominator, we use the time for the minimum number of nodes used, which in this case equates to the total time required to complete a GAMESS job on 8 nodes running a single process per node.

t_N=Total run time of GAMESS on a given number of nodes.
t_1=Total run time of GAMESS for a single node.

$$Scalability = \frac{1}{\frac{t_N}{t_1}}$$

TABLE II
AT HF-SCF CONVENTIONAL ON NIAGARA T2 PROCESSOR

Cores / Procs Per Core	1	2	4	8
1	X	1	1.89	3.32
2	1	1.88	3.34	X
4	1	1.75	X	X
8	1	X	X	X

TABLE III
AT HF-SCF DIRECT ON NIAGARA T2 PROCESSOR

Cores / Procs Per Core	1	2	4	8
1	X	X	1	1.93
2	X	1	1.93	X
4	1	1.94	X	X
8	1	X	X	X

Consider the results shown in Tables II, III for the Niagara machine. Since the Niagara machine consists of a single node and 8 cores, the scalability is calculated with respect to the increase in the number of cores. We can see that as we increase the number of cores and keep the number of processes per core constant, the scalability increases similarly. The scalability is better than Franklin. However, the runtimes are slower than Franklin mainly due to the sharing of the L2 Cache and the I/O channel between the cores. Franklin processors have access to a bigger cache per core and a faster clock rate. Also, the *direct* mode of operation shows better scalability than the *conventional* due to the increase in the IO contention as the number of processes increase. We also observed that if we increase the number of threads and keep the number of processes constant, the runtimes stabilize once the number of threads are equal to the number of cores on which they are run. Even if we increase the number of cores, the performance remains flat. Thus the best possible combination for running the GAMESS application would be to allocate individual threads to single cores.

The Tables IV and V, give the scalability results for Franklin. For the *conventional* run of AT on Franklin, the scalability improves by 60% when we move from 4 nodes to 8 nodes but a quadruple increase in the number of nodes (from 4 to 16) does not improve the performance 4 times. The scalability is only 2.2 times in this case. Also, the scalability actually reduces from 1.83 to 1.64 when we use 8 nodes to run 4 processes each instead of 4 nodes. Since AT is a small molecule, running 32 processes increases the communication delay leading to reduction in scalability. For

TABLE IV
AT SCF CONVENTIONAL ON FRANKLIN NERSC CLUSTER

Nodes Procs Per Node	2	4	8	16
1	X	X	1	1.66
2	X	1	1.59	2.22
4	1	1.83	1.64	X

TABLE V
AT SCF DIRECT ON FRANKLIN NERSC CLUSTER

Nodes Procs Per Node	2	4	8	16
1	X	X	1	1.86
2	X	1	1.95	3.42
4	1	1.77	3.08	X

TABLE VI
AT MP2 CONVENTIONAL ON FRANKLIN NERSC CLUSTER USING COMMUNICATION TIME

Nodes Procs Per Node	2	4	8	16
1	X	X	1	1.98
2	X	1	1.87	3.45
4	1	1.64	2.57	X

TABLE VII
AT MP2 DIRECT ON FRANKLIN NERSC CLUSTER USING COMMUNICATION TIME

Nodes Procs Per Node	2	4	8	16
1	X	X	1	1.97
2	X	1	1.84	3.4
4	1	1.64	2.54	X

the *direct* job execution of AT, the scalability looks good and the performance nearly doubles when the number of nodes is doubled. The quadrupling of the nodes does not have the same effect though it does provide a three time increase in the performance.

The Tables VI and VII, give the scalability results for Franklin for MP2 computations. The results are based only on the communication time measured using TAU. The default MP2 option (CODE=DDI in the input file) uses remote memory access to store the intermediate results. This causes the spike in the communication time measured with TAU [15]. The communication routines take about 90% of the total time for *conventional* implementation and about 70% of the total time for the *direct* implementation. We can see from the tables that the scaling for the communication time is constant for both *conventional* and *direct* implementation. Comparing the scalability data for SCF and MP2, we see an anomaly at 8X4 (4 processes each on 8 nodes) for the *conventional* execution of SCF. There is a slowdown at 8X4 and this is due to an increase in the communication time percentage. AT is a small molecule and the usage of 32 processes over 8 nodes causes congestion leading to a drop in the execution performance. Such anomaly detection through the scalability analysis module can be done for all molecules using a combination of communication, computation, IO and Total time.

These tables give us a good idea about the scalability on different machines through the usage of the total time taken by GAMESS jobs. We could use communication time, IO time or computation time instead of total time and get the same scalability analysis done. Also, this scalability computation may be integrated with NICAN to provide NICAN with scalability information for either monitoring or adaptation.

VI. CONCLUSION

The performance characteristics of quantum chemistry applications such as GAMESS is determined by its numerous input parameters and the architecture on which the application is executed. We conducted performance tests for different molecule sets on multiple architectures. HF-SCF and MP2 computations for *direct* and *conventional* implementations were tested. The data analysis was made simpler by the use of a *PostGreSql* database to store the huge volume of the performance data. The obtained performance data was analyzed and a set of rudimentary and simple rules were created to augment the existing adaptation process. A database framework was created around NICAN so as to access the *PostGreSql* database and aid GAMESS adaptation. The performance data is collected offline and then inserted into the database. The existing dynamic adaptation of GAMESS has not been modified. The performance improvement obtained by using the database framework has been shown up to 40%. A scalability analysis module has also been created which calculates the scalability of each architecture as a medium to run GAMESS jobs.

It is not possible to store performance data and rules for all the molecules used by the chemists. Hence, as a future work, we envision that NICAN should be able to recognize the similarity of the given input molecule with the data available in the performance database and then approximate the required data. Application related data such

as compiler options or system related data such as cache performance data may be extracted from the application execution and stored in the database. This can help us to refine the adaptation strategy. The performance data has been collected on different architectures. Each architecture has its unique features but they also have features which can be compared. Using the performance data, we intend to compare different architectures and also extrapolate the performance of GAMESS on other architectures. The granularity of the performance data can be further increased so that we can get the data within a particular computational phase or even within an individual SCF iteration. The NICAN dynamic adaptation algorithm can be modified to use communication time or IO time, instead of just the iteration time to decide on the switch between *conventional* and *direct*. The dynamic adaptation algorithm is currently in use only for the SCF iterations. MP2 has different implementations that can be switched dynamically. Also, the DFT (Density Functional Theory) implementation can be modified to use dynamic adaptations.

REFERENCES

[1] J. Dongarra and V. Eijkhout Self-adapting Numerical Software for Next Generation Applications International Journal of High Performance Computing applications (IJHPCA) volume 17, Number 2, Pg 125-131, 2003

[2] K. Douglas and S. Douglas *PostgreSQL* , 2003 New Riders Publishing,Thousand Oaks, CA, USA,

[3] D.Feller The role of databases in support of computational chemistry calculations Journal of Computational Chemistry volume 17, Number 13, Pg 1571-1586, 1996

[4] H. Liu and M. Parashar Enabling self-management of component-based high-performance scientific applications HPDC '05: Proceedings of the High Performance Distributed Computing, 2005. HPDC-14. Proceedings. 14th IEEE International Symposium Pg 59–68, 2005

[5] F. Jensen Introduction to Computational Chemistry, Wiley, Chester, UK, 1999.

[6] C.L. Janssen, I.B. Nielsen, M.L. Leininger, E.F. Valeev, J.P. Kenny, E.T. Seidl . The Massively Parallel Quantum Chemistry Program (MPQC), 3.0, Sandia National Laboratories, Livermore, CA, USA, 2008.

[7] R. Kendall, E. Aprà, D. Bernholdt, E. Bylaska, M. Dupuis, G. Fann, R. Harrison, J .Ju, J .Nichols, J. Nieplocha, T.P. Straatsma, T. Windus and A. Wong High performance computational chemistry: An overview of NWChem a distributed parallel application In Journal of Computer Physics Communications, June 2000, No 1-2, Pages 260-283, Volume 128

[8] P. Kongetira, K. Aingaran and K. Olukotun A 32-way Multithreaded SPARC(R) Processor. In IEEE Micro, Volume 25, Number 2, 2005, Pg 21–29.

[9] D. Kulkarni and M. Sosonkina. A framework for integrating network information into distributed iterativesolution of sparse linear systems. High Performance Computing for Computational Science - VECPAR 2002, 5th International Conference, Porto, Portugal, June 26-28, 2002, Selected Papers and Invited Talks, volume 2565 of Lecture Notes in Computer Science, pages 436–450. Springer, 2003.

[10] L. Li, J. Kenny, M.S. Wu, K. Huck, A. Gaenko, M.S. Gordon, C.L. Janssen, L.C. McInnes, H. Mori, H.M. Netzloff, B. Norris, T.L. Windus. Adaptive Application Composition in Quantum Chemistry Proceedings of The 5th International Conference on the Quality of Software Architectures (QoSA 2009) February 2009

[11] R. Olson, M.W. Schmidt, M.S. Gordon and A.Rendell Enabling the Efficient Use of SMP Clusters: The GAMESS/DDI Model, Proceedings of the 2003 ACM/IEEE conference on Supercomputing, p.41, November 15-21, 2003.

[12] M. Schmidt, K. Baldridge, J. Boatz, S. Elbert, M.S. Gordon, J. Jensen, S. Koseki, N. Matsunaga, K.Nguyen, S. Su, T. Windus, M. Dupuis, J. Montgomery,Jr. General Atomic and Molecular Electronic Structure System. *Journal of Computational Chemistry*, 14, 1347-1363(1993).

[13] Schuchardt, Karen L. and Didier, Brett T. and Elsethagen, Todd and Sun, Lisong and Gurumoorthi, Vidhya and Chase, Jared and Li, Jun and Windus, Theresa L. Basis Set Exchange: A Community Database for Computational Sciences.*Journal of Chemical Information and Modeling*, Volume 47, Number 3, Pg 1045–1052(2007).

[14] L. Seshagiri, M. Sosonkina, Z. Zhang Electronic Structure Calculations and Adaptation Scheme in Multi-core Computing Environments In Proceedings of 2009 International Conference on Computational Science (ICCS-2009), Baton Rouge, Louisiana, May 25-27, 2009 Pg 3–12, Lecture Notes In Computer Science; Vol. 5544, Springer-Verlag

[15] L. Seshagiri, M.S. Wu, M. Sosonkina, Z. Zhang Exploring Tuning Strategies for Quantum Chemistry Applications In Proceedings of 2009 International Workshop on Automatic Performance Tuning (iWAPT-2009), Tokyo, Japan, Oct 1-2, 2009

[16] S. Shende and A. Malony The TAU parallel performance system Int. J. High-Perf. Computing Appl., ACTS Collection special issue 20 (Summer 2006), Pg 287-331.

[17] M. Sosonkina. Adapting Distributed Scientific Applications to Run-time Network Conditions. In Applied Parallel Computing, State of the Art in Scientific Computing, 7th International Workshop, PARA 2004, Revised Selected Papers, volume 3732 of Lecture Notes in Computer Science, pages 745–755. Springer, 2006.

[18] M. Sosonkina, S. Storie. Parallel performance of an iterative method in cluster environments: an experimental study. In *Proceedings of Parallel Matrix Algorithms and Applications (PMAA 2004)*, Marseille, October 2004.

[19] S. Storie. Aspects of Communication Subsystem Analysis for Distributed Scientific Applications. Master's Thesis, University of Minnesota Duluth, May 2004.

[20] Sun Microsystems Inc. http://www.sun.com/processors/UltraSPARC-T2/.

[21] H.-J. Werner, P. J. Knowles, R. Lindh, F. R. Manby, M. Schütz and others. MOLPRO, version 2008.3, a package of ab initio programs

[22] N. Ustemirov, M. Sosonkina, M.S. Gordon and M.W. Schmidt Dynamic Algorithm Selection in Parallel GAMESS Calculations. ICPPW '06: Proceedings of the 2006 International Conference Workshops on Parallel Processing 2006, Pg 489–496

[23] N. Ustemirov, M. Sosonkina, M.S. Gordon, M.W. Schmidt. Concurrent Execution of Electronic Structure Calculations in SMP Environments. In Proceedings 2005 Spring Simulation MultiConf, High Performance Computing Symposium, J.A.Hamilton Jr. et al, Soc. for Modeling and Simulation Internat. , San Diego, CA. 2005

[24] M.S. Wu, J.L. Bentz, F. Peng, M. Sosonkina, M.S. Gordon, R.A. Kendall Integrating Performance Tools with Large-Scale Scientific Software. In Proceedings of IEEE International Parallel and Distributed Processing Symposium, 2007. (IPDPS 2007). Pg 1–8

[25] Y. Shao, L. Fusti-Molnar, Y. Jung, J. Kussmann, C. Ochsenfeld, S. T. Brown, A. T. B. Gilbert, L. V. Slipchenko,S. V. Levchenko, D. P. O'Neill, R. A. Distasio Jr., R. C. Lochan, T. Wang, G. J. O. Beran, N. A. Besley, J. M., Herbert, C. Y. Lin, T. Van Voorhis, S. H. Chien, A. Sodt, R. P. Steele, V. A. Rassolov, P. E. Maslen, P. P. Korambath, R. D. Adamson, B. Austin, J. Baker, E. F. C. Byrd, H. Dachsel, R. J. Doerksen, A. Dreuw, B. D. Dunietz, A. D. Dutoi, T. R. Furlani, S. R. Gwaltney, A. Heyden, S. Hirata, C.-P. Hsu, G. Kedziora, R. Z. Khalliulin, P. Klunzinger, A. M. Lee, M. S. Lee, W. Liang, I. Lotan, N. Nair, B. Peters, E. I. Proynov, P. A. Pieniazek, Y. M. Rhee, J. Ritchie, E. Rosta, C. D. Sherrill, A. C. Simmonett, J. E. Subotnik, H. L. Woodcock III, W. Zhang, A. T. Bell, A. K. Chakraborty, D. M. Chipman, F. J. Keil, A. Warshel, W. J. Hehre, H. F. Schaefer III, J. Kong, A. I. Krylov, P. M. W. Gill, M. Head-Gordon Advances in methods and algorithms in a modern quantum chemistry program package. Phys. Chem. Chem. Phys., (2006) 8, 3172 - 3191.

[26] T.D. Crawford, C.D. Sherrill, E.F. Valeev, J.T. Fermann, R.A. King, M.L. Leininger, S.T. Brown, C.L. Janssen, E.T. Seidl, J.P. Kenny, and W.D. Allen, PSI3: An Open-Source Ab Initio Electronic Structure Package IJ. Comp. Chem. 28, 1610-1616 (2007).

[27] G. Karlström, R. Lindh, P.-Å. Malmqvist, B.O. Roos, U. Ryde, V. Veryazov, P.-O. Widmark, M. Cossi, B. Schimmelpfennig, P. Neogrady, and L. Seijo. Molcas: a program package for computational chemistry Computational Material Science, Vol 28, Pg222, 2003.